制約システムとストリームを使った仮想的なマリオネットの表現

An Expression of Virtual Marionette with a Constraint System and Streams

コールドブレス有限会社 Cold Breath, Inc.
久井 亨 Toru HISAI

概要

プロジェクト *MEPHISTO* は、チェコ式のマリオネットの構造から着想を得て、制約システムと無限ストリームを使った新しい 3 次元モデルのアニメーションの表現を提案する。モデルを階層を持たないフラットな構造で表現し、環境や内部状態の時間変化をストリームを使ってアクセスすることにより、モデルと環境の相互作用をより簡単で自然に表現できる。これにより、テレビゲーム等の開発がより効率的になることが期待できる。

Project *MEPHISTO* suggests a new expression of 3-D animation with a constraint system and streams, inspired by Czech marionette. Models are expressed in a flat data structure without hierarchy and changes of environment and internal states are accessed by streams, so that expression of interaction between models and environment becomes easier and natural. It's hoped that we will be able to develop software such as computer games effectively.

1 マリオネットと *MEPHISTO*

マリオネット（marionette）とは、チェコなどで古くから伝わる人形劇のための糸操り人形である。木や粘土などで作られたパーツを皮ひもなどを使ってつなぎ合わせ、7-9 本程度の糸を一部のパーツに結び付けて操作出来るようになっている。

多くのマリオネットは 10 本に満たない少ない糸で操作できるようになっているが、各パーツの間にある力学的、運動学的な相互作用によって、まるで生きているかのような表情豊かな動作が可能である。

ここで提案するプロジェクト「*MEPHISTO*（メフィスト）」は、チェコ式の糸操りのマリオネットからヒントを得て、その動きを制約システムを使って再現しようという試みである。

制約システムを使うことで、操作に必要なパラメタを少なく抑えられるため、ゲームのようなインタラクティブな操作環境でも、簡単な操作で多彩な動作が実現出来ると考えられる。

ところで、チェコの人形劇にも馴染みの深い戯曲に「ファウスト」がある。このプロジェクトのコード名は、「ファウスト」に登場する悪魔メフィストフェレスに因んだものである。

2 制約システムを使ったモデルの表現

2.1 従来の方法

テレビゲーム等に登場するキャラクターのモデルは、図 1 のようにそれを構成するパーツをノードとする木構造で表現することが多い。それぞれのノードには座標変換行列を割り当て、ノードごとに局所座標系をもつ。これは、腕や脚などの複数の関節を持つモデルを表現するには自然な方法である。

しかし、手先のような末端の座標を求めるためには、ルートノードから手先のノードに至るまでの各ノードについて、そのノードに割り当てられた座標変換行列の積を計算しなくてはいけない。また、手先の座標をあたえて、肘や肩の角度を求めるためには、**逆運動学**的な計算が必要となる。

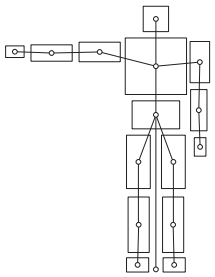

図 1　木構造のモデル

2.2　局所座標系を捨て、全てを制約で制御する

MEPHISTO では従来の木構造を捨て、全てのノードをフラットに扱う。ノードとノードのつながりは、**制約システム**を使って記述する。

ただし、ここで現れるノードは、必ずしもパーツのオブジェクトだけではなく、そのモデルを特徴づける各種の方向ベクトルや、その他のモデル固有のパラメタをも含ませることができる。

2.2.1　例：2 本足ロボット

ここで例として、2 本の脚からなるごく簡単な歩行ロボットのモデルを作成する。

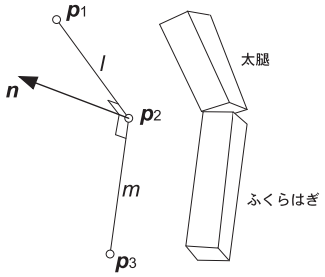

図 2　片方の脚の構造

まず、左右のそれぞれの脚は、図 2 のように 3 個の点 $\boldsymbol{p}_1$、$\boldsymbol{p}_2$、$\boldsymbol{p}_3$ と、方向ベクトル $\boldsymbol{n}$ を使って表現する。

$\boldsymbol{p}_1$、$\boldsymbol{p}_2$、$\boldsymbol{p}_3$ はそれぞれ、脚の付け根、膝、足首の点であり、これらの点の間に太腿とふくらはぎのパーツが描画されることになる。また、$\boldsymbol{n}$ は $\boldsymbol{p}_1$、$\boldsymbol{p}_2$、$\boldsymbol{p}_3$ を含む平面の法線ベクトルである。太腿とふくらはぎの長さをそれぞれ l、m とすれば、次のように書くことができる。

$$|\boldsymbol{p}_2 - \boldsymbol{p}_1|^2 = l^2 \tag{1}$$

$$|\boldsymbol{p}_3 - \boldsymbol{p}_2|^2 = m^2 \tag{2}$$

$$\frac{(\boldsymbol{p}_3 - \boldsymbol{p}_2) \times (\boldsymbol{p}_2 - \boldsymbol{p}_1)}{|(\boldsymbol{p}_3 - \boldsymbol{p}_2) \times (\boldsymbol{p}_2 - \boldsymbol{p}_1)|} = \boldsymbol{n} \tag{3}$$

この関係を、脚のモデルについての制約 C_1 として設定すると、図 3 のような制約システムができあがる。

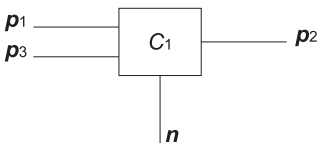

図 3　片方の脚の制約システム

次に、図 4 のようにして 2 本の脚を股関節に接続する。

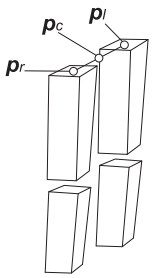

図 4　2 本の脚を接続

ここで改めて、左右の脚の付け根の点をそれぞれ $\boldsymbol{p}_l$、$\boldsymbol{p}_r$、法線ベクトルをそれぞれ $\boldsymbol{n}_l$、$\boldsymbol{n}_r$ とおき、$\boldsymbol{p}_l$、$\boldsymbol{p}_r$ の中点（腰の位置）を $\boldsymbol{p}_c$ とすると、次の関係が成り立つ。

$$\boldsymbol{p}_l - \boldsymbol{p}_c = \boldsymbol{p}_c - \boldsymbol{p}_r \tag{4}$$

$$\frac{\boldsymbol{p}_l - \boldsymbol{p}_c}{|\boldsymbol{p}_l - \boldsymbol{p}_c|} = \boldsymbol{n}_l = \boldsymbol{n}_r \tag{5}$$

この関係を制約 C_2 とすると、制約システムは図 5 のようになる。

以上の関係を、制約システムの中に組み入れることで、パーツ間の幾何的な関係を記述する。

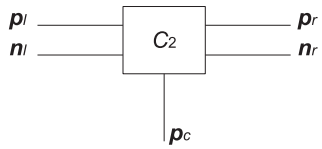

図 5　股関節の制約システム

ただし、全てのパーツの状態を決定するには、上記の制約だけでは不十分である。なぜなら、実際のアニメーションでは、そのキャラクタの姿勢や動作の種類によって、動的に制約が変化するが、上記の制約は、このモデルが常に従うべきものだけを記述しているのに過ぎないからである。

次の節では、時間軸に沿って変化するパラメタを制約システムに組み込み、モデルのアニメーションを実現している方法を説明する。

3　無限ストリームによる時間の表現

ここでいったん制約のことからは離れて、アニメーションにおける時間について考える。

3.1　時間とは何か

アニメーションとは、たくさんの静止画を人間の目では認識できない位の速さで次々に映すことで、あたかもそこに描かれている絵が動いているかのように見せる手法である。

例えば映画であれば、細長いフィルムに一定の時間間隔 Δt ごとに映すべき静止画が焼かれている。この静止画の 1 枚 1 枚を**フレーム**と呼ぶ。各フレームにはそれぞれ映される時刻が結びつけられているが、その時刻とは映写開始時刻を T_0 とすると、ある整数 n が存在して $T_0 + n\Delta t$ で表される。したがって、映画における時刻は、この整数 n と同一視することができる。

3.2　無限ストリーム

$\mathcal{MEPHISTO}$ では、時刻 n と描画手続き $proc$ の対を使って、次のような形式でフレームを表現する。

$$(n\ .\ proc) \tag{6}$$

さらに、アニメーションをフレームの無限**ストリーム**で表現する。無限ストリームとは無限の長さを持つリストである。概念的には、次のような構造の無限リストとして扱うことができる。

$$((1\ .\ p_1)(2\ .\ p_2)(3\ .\ p_3)\ \ldots) \tag{7}$$

ストリームの特徴として、ある要素の値から次の要素の値を決定することができる。つまり (7) の形式でいえば、p_n は p_{n-1} の関数であってよい。ストリームはこの性質のため、入力に対してリアルタイムに反応するような、対話的なアニメーションを表現するのに適しているのである。

3.2.1　ストリームの効用

ストリーム同士の重ね合わせや連結などの演算も簡単に定義できるため、別々に作った複数のモデルのアニメーションを、複雑に絡めあわせることも比較的容易に実現できる。

ゲームの場合、各キャラクタの内部状態や各種イベントに関するフラグ、デバイスの入力状態など、さまざまなタイプの状態がそれぞれ違った形式で表現されることが多いが、これはプログラマにとって非常に扱いにくい。これらの状態も、ストリームを使うと統一的に表現し、アクセスすることができる。また、ストリームを使えば、副作用のない関数的なインタフェイスを作ることができる。

これは、短期間で効率的な開発を行う上でも重要な要素となると考えられる。

3.2.2　例：2 本足ロボットの続き

ここで、ストリームを使って 2.2.1 節で定義した 2 本足ロボットを歩かせてみる。詳細な説明は煩雑になるので割愛するが、だいたい以下のような流れでアニメーションを定義することができる。

歩行動作のアニメーションを定義する方法はいろいろと考えられるが、ここでは図 6 のように左右の足が地面を踏む位置を次々に与えることで、この動作を実現する。

左右の足首の軌道は、地面を踏む点を放物線のような曲線で補間することで得られる。さらに、この軌道上の各点に時刻を結びつけることで、足首の位置の時間変化を定義することができる。

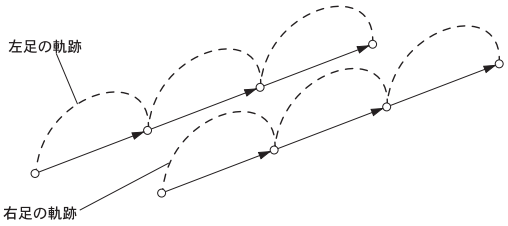

図 6　左右の足の軌道

左右の足首の位置 $\boldsymbol{p}_{F1}$、$\boldsymbol{p}_{F2}$ と鉛直上向きの方向ベクトル $\boldsymbol{u}$ によって、腰の位置 $\boldsymbol{p}_c$ と進行方向ベクトル $\boldsymbol{f}$ が求まるので、図 7 のような制約システムが与えられる。

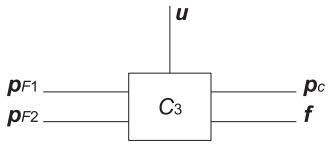

図 7　歩行中の制約システム

最後に、時間の経過に伴って左右の足の位置を更新する手続きのストリームを作れば、既に定めた制約によって各時刻における両脚の各点が求まるので、歩行のアニメーションが実現する。

ここでは足が地面を踏む位置はあらかじめ与えたが、これをデバイスからの入力を元に随時計算するようにすれば、このロボットを簡単な操作で自由に歩かせることができる。

さらに、足が地面につく高さに変化を持たせれば、起伏のある地形も簡単に表現できる。

4　実装

MEPHISTO は、Scheme 処理系 **Gauche** と、その OpenGL バインディングである **Gauche-gl** を用いて記述されている。制約システムの処理には [2] を参考に開発したごく簡単な制約解消系を使い、ストリームの処理には Gauche に付属のストリームライブラリを用いた。また、実装方法のアイディアのほぼ全てを [1] に依った。

ただし、このプロジェクトの趣旨はあくまでも表現方法の提案であり、実用化にあたっては、必要に応じて他の環境への移植も検討する。

5　終わりに：マリオネットと制約

糸操りのマリオネットは全ての操作を「糸」を隔てて行うため、棒操り人形やハンドパペットなどの他の劇人形と違って、パーツを直接的に制御することができない。このため一見不自由にも見えるが、逆に糸のおかげで人形の動きに滑らかさが生まれ、また、舞台装置の間のちょっとした差違がうまく吸収されるという利点を持つ。

そこで *MEPHISTO* では、制約という「糸」を使ってパーツの多くの部分を間接的に操作することで、自然な動きを生み出そうという狙いがある。

ゲームではしばしば、「銃を敵に向けながら走る」というような、複合的な動作をさせたいことがある。しかし従来のように、ひとつひとつのモーションを関節の角度を制御することで表現する場合、上半身と下半身のモーションを組み合わせるなどということが困難である。

また、モーションのひとつひとつは確かに人の手で丁寧に作られていても、ゲーム画面中でそれが単調に繰り返されてしまうと、どうしても機械的に見えてしまう。

制約とストリームを使ったモーションの定義は、このような動きのバリエーションの困難さを解決する、ひとつの手段になり得ると考えている。

なお、このプロジェクトは IPA による 2005 年度未踏ユースに採択され、さまざまな形で支援をいただいています。*MEPHISTO* についての最新情報は随時ホームページ [3] にて公開しています。

参考文献

[1] ジェラルド・ジェイ・サスマン他『計算機プログラムの構造と解釈 第二版』ピアソン・エデュケーション、2000 年

[2] Mark Jason Dominus, *Higher-order Perl: A Guide To Program Transformation*, Morgan Kaufmann Pub, 2005 年

[3] *MEPHISTO* プロジェクトホームページ：`http://mephisto.torus.jp/`